Arrays – Walkthrough

Passing Arrays to Methods

To pass an array argument to a method, specify the name of the array without using brackets

For example, if array hourlyTemperatures declared as

**int[] hourlyTemperatures = new int[ 24 ];**

the method call

**ModifyArray(hourlyTemperatures);**

passes array hourlyTemperatures to method ModifyArray

Every array object “knows” its own size (via the Length instance variable), so when we pass an array object into a method, we do not pass the size of the array as an argument separately

Although entire arrays are passed by reference, individual array elements of primitive data types are passed by value, the same way as simple variables are

(The objects referred to by individual elements of a nonprimitive-type array are still passed by reference)

Such simple single pieces of data are sometimes called scalars or scalar quantities

To pass an array element to a method, use the subscripted name of the array element as an argument in the method call

For a method to receive an array through a method call, the method’s parameter list must specify that an array is to be received

For example, the method header for method ModifyArray might be written as

**public void ModifyArray(int[] b)**

indicating that ModifyArray expects to receive an integer array in parameter b

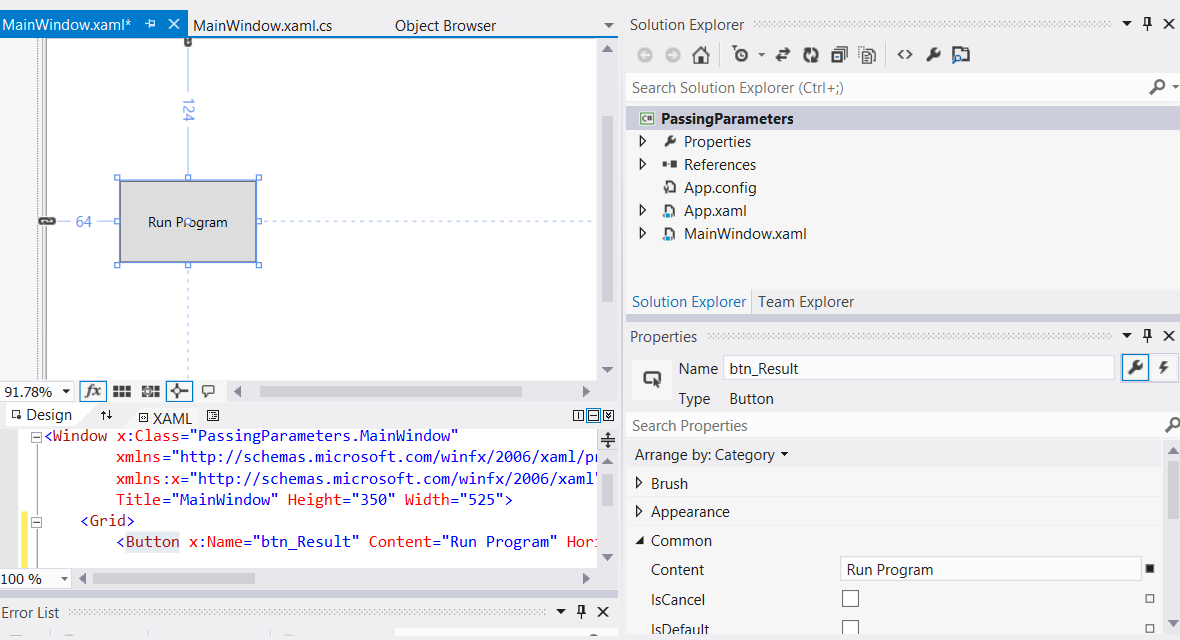
Arrays are passed by reference; when the called method uses the array name b, it refers to the actual array in the caller (array hourlyTemperatures)

We are going to build a basic application which has a single button, which when clicked will:

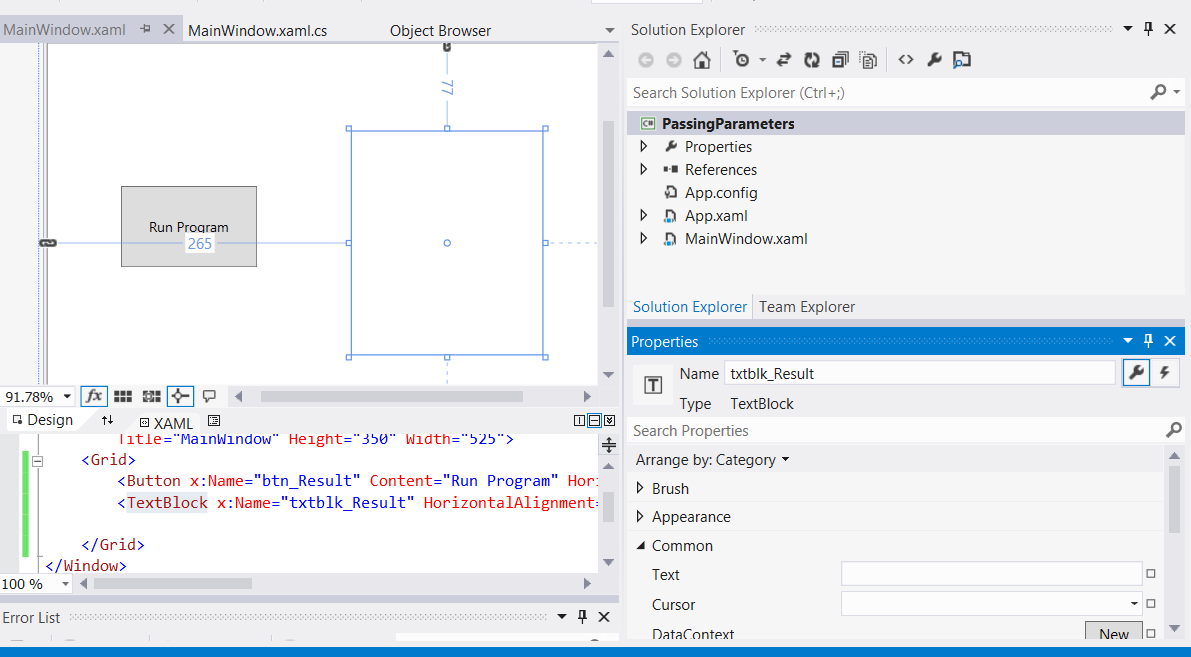
* Display the contents of an array before and after it is passed-by-reference and modified
* Display the value of an array element before and after it is passed-by-value and modified

The aim of this tutorial is to show how to pass entire arrays (by reference) and single array elements (by value, assuming it is a value type)

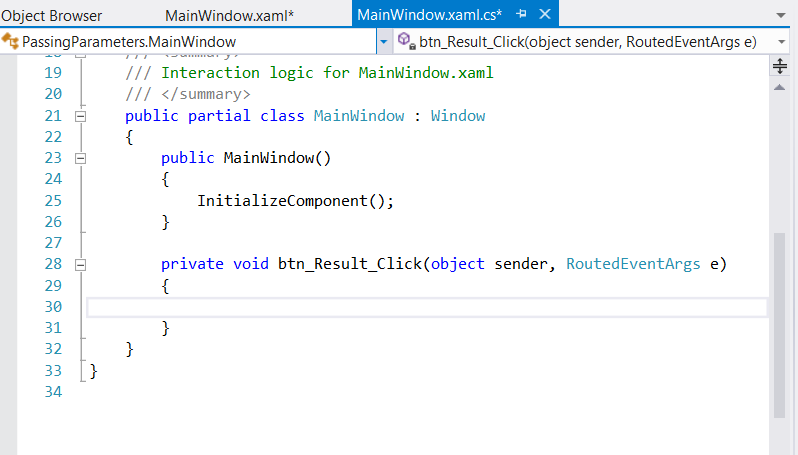
1. Create a new WPF project
2. Drag across a button from the toolbox
   1. set the “Name” property to btn\_Result
   2. set the “Content” property to “Run Program”



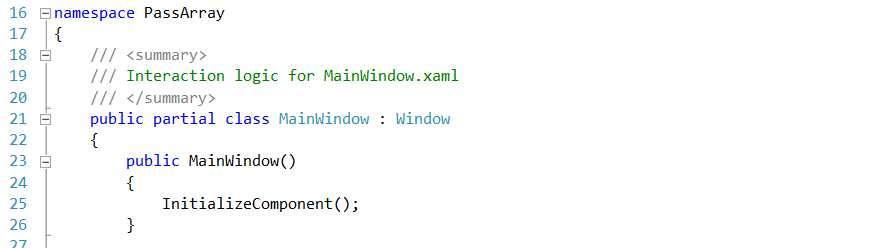
1. Drag across a text block and from the toolbox
   1. set the “Name” property to txtblk\_Result
   2. set the “Text” property to blank

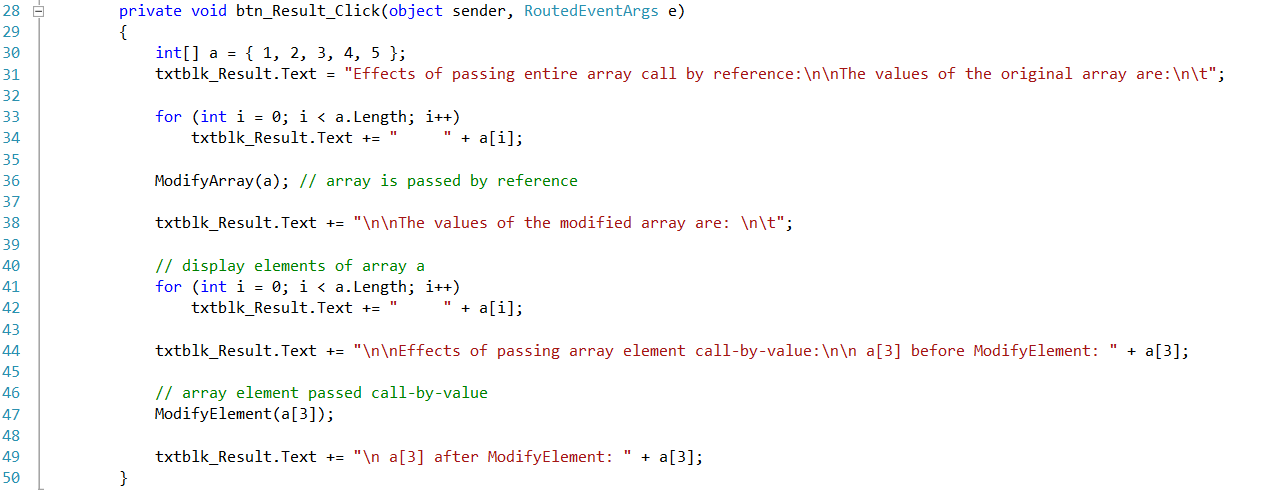


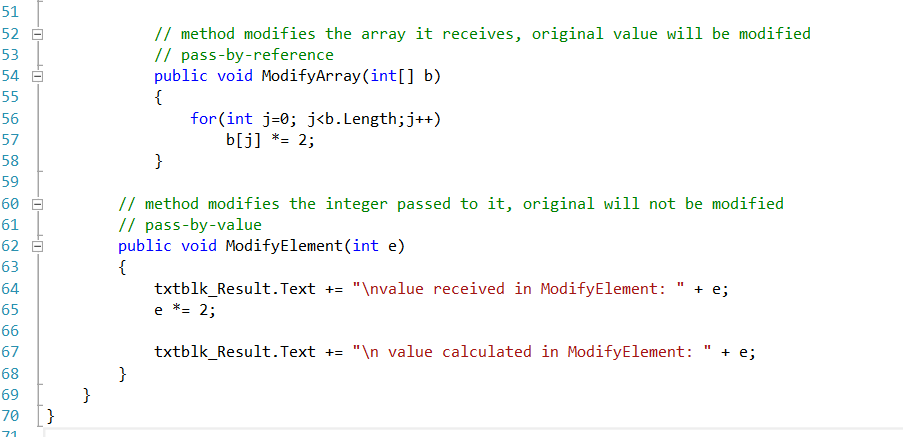
1. Double-click on the “Run Program” button to automatically create an event handler in the MainWindow.xaml.cs file, which is a method called btn\_Result\_Click



1. We are now going to write the code for this event handler and additional “helper” methods called “ModifyArray” and “ModifyElement”. The final code can be viewed in the screenshot below followed by a full explanation







Passing arrays and individual array elements to methods - code walkthrough

The for loop on lines 33–34 appends the five elements of integer array a to the Text property of txtblk\_Result

Line 36 invokes method ModifyArray and passes to it array a

Method ModifyArray multiplies each element by 2

To illustrate that array a’s elements were modified, the for loop on lines 41 - 42 appends the five elements of integer array a to the Text property of txtblk\_Result

To show the value of a[ 3 ] before the call to ModifyElement, line 44 appends the value of a[ 3 ] (and other information) to txtblk\_Result.Text

Line 47 invokes method ModifyElement and passes a[ 3 ]

Remember that a[ 3 ] is a single int value in the array a

Also, remember that values of primitive types always are passed to methods by value

Therefore, a **copy** of a[ 3 ] is passed

Method ModifyElement multiplies its argument by 2 and stores the result in its parameter e

The parameter of ModifyElement is a local variable, so when the method terminates, the local variable is destroyed

Thus, when control is returned to PassArray, the unmodified value of a[ 3 ] is appended to the txtblk\_Result.Text (line 49)